# DESIGN AND VERIFICATION OF UART COMMUNICATION PROTOCOL USING SYSTEM VERILOG

**A PROJECT REPORT**

*Submitted by*

**Umesh Kanna. K. B (2022105010)**

**Rubesh. K (2022105702)**

**Jaganath. B (2022105522)**

*In partial fulfillment for the award of the degree*

*of*

**BACHELOR OF ENGINEERING**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

**COLLEGE OF ENGINEERING GUINDY,**

**ANNA UNIVERSITY: CHENNAI 600 025**

**NOVEMBER 2025**

# BONAFIDE CERTIFICATE

Certified that this project report **“DESIGN AND VERIFICATION OF UART COMMUNICATION PROTOCOL USING SYSTEM VERILOG”** is the Bonafide work of “**Umesh Kanna. K. B** (2022105010), **Rubesh. K** (2022105702), **Jaganath. B** (2022105522)” who carried out the project work under my supervision.

| | |
|---|---|
| **SIGNATURE** | **SIGNATURE** |
| **Dr. M. A. BHAGYAVENI** | **Dr. S. R. SRIRAM** |
| **HEAD OF THE DEPARTMENT** | **SUPERVISOR** |
| PROFESSOR | ASST PROFESSOR |
| Department of Electronics and Communication Engineering | Department of Electronics and Communication Engineering |
| College of Engineering Guindy, | College of Engineering Guindy, |
| Anna University, | Anna University, |
| Chennai – 600 025 | Chennai – 600 025 |

## ACKNOWLEDGEMENT

We express our wholehearted gratitude to our dynamic and zestful Head of the Department Dr. M. A. BHAGYAVENI, Professor, Department of Electronics & Communication Engineering for her continued and heartening support throughout the project.

We manifest our sincere thanks and regards to our project supervisor, Dr. S. R. SRIRAM, Professor, Department of Electronics and Communication Engineering for her ardent support, patience, valuable guidance, technical expertise and encouragement in our project.

We also are supremely grateful to the teaching and non-teaching staff of the Department of Electronics and Communication Engineering, for their kind help and co-operation during the course of our project.

Last but not the least, we would not have made this project without the support from our beloved family and friends.

## ABSTRACT

This project presents the design and verification of a Universal Asynchronous Receiver Transmitter (UART) communication protocol using System Verilog. UART is one of the most fundamental serial communication interfaces, widely used for reliable, low-speed data transfer between digital systems. In this work, both the transmitter and receiver modules are designed with parameterized baud rate and clock frequency to ensure flexibility and scalability. The main objective is to verify the end-to-end functionality of UART communication using a SystemVerilog-based, UVM-like verification architecture.

The verification environment is developed with modular components such as Generator, Driver, Monitor, Scoreboard, and Interface, all connected through mailboxes to enable synchronized data flow and automated checking. The Generator creates UART transactions, the Driver applies them to the DUT, the Monitor captures received data, and the Scoreboard compares expected and actual results to declare pass/fail status. This self-checking environment minimizes manual intervention and ensures thorough validation of the design. Simulations are executed on EDA Playground, with both waveform and console outputs confirming correct data transmission and reception. The results demonstrate successful functional verification of the UART protocol, establishing a reusable and scalable verification framework for future communication IP designs.

## TABLE OF CONTENTS

| CHAPTE R NO. | TITLE | PAGE NO. |
|---|---|---|
| | **ABSTRACT** | 3 |
| **1** | **INTRODUCTION** | 5 |
| **2** | **BLOCK DIAGRAM** | 6 |
| **3** | **LITERATURE REVIEW** | 7 |
| **4** | **PROPOSED WORK** | 9 |
| **5** | **UART CODE** | 11 |
| | DESIGN CODE | 11 |
| | TESTBENCH CODE | 14 |
| **6** | **OUTPUT** | 18 |
| **7** | **REFERENCES** | 19 |

# CHAPTER 1
# INTRODUCTION

The Universal Asynchronous Receiver Transmitter (UART) is one of the simplest and most widely used serial communication protocols in digital systems. It enables reliable data exchange between devices by transmitting data serially, bit by bit, without requiring a clock signal. UART is extensively used in embedded systems, microcontrollers, and FPGAs due to its simplicity, low hardware cost, and ease of implementation. A typical UART communication consists of two main modules — the transmitter and receiver — that convert parallel data into serial format and vice versa.

The UART transmitter adds start and stop bits to each data byte before transmission, while the receiver detects these bits, samples the incoming serial data, and reconstructs the original parallel data. Since UART communication is asynchronous, both ends must operate at the same baud rate to avoid data loss. Verifying the correct operation of these modules is essential to ensure reliable data transfer, especially in real-time systems where timing and synchronization are critical.

In this project, the UART transmitter and receiver design are verified using **SystemVerilog** through a **UVM-like testbench architecture**. SystemVerilog provides object-oriented features that make verification modular, reusable, and scalable. The testbench includes key components such as the generator, driver, monitor, and scoreboard, which communicate via mailboxes. The generator produces input data transactions, the driver applies them to the DUT (Design Under Test), the monitor observes the DUT outputs, and the scoreboard compares expected and actual results to ensure correct UART functionality.

The main objective of this work is to validate the UART's transmit and receive operations under different test scenarios using a structured verification approach. By employing a self-checking testbench, the design ensures automated comparison of data and provides clear pass/fail indications. This verification environment not only confirms the correctness of the UART design but also demonstrates the effectiveness of a modular SystemVerilog-based verification methodology, which can be extended to other communication protocols in future developments.

# CHAPTER 2

# BLOCK DIAGRAM

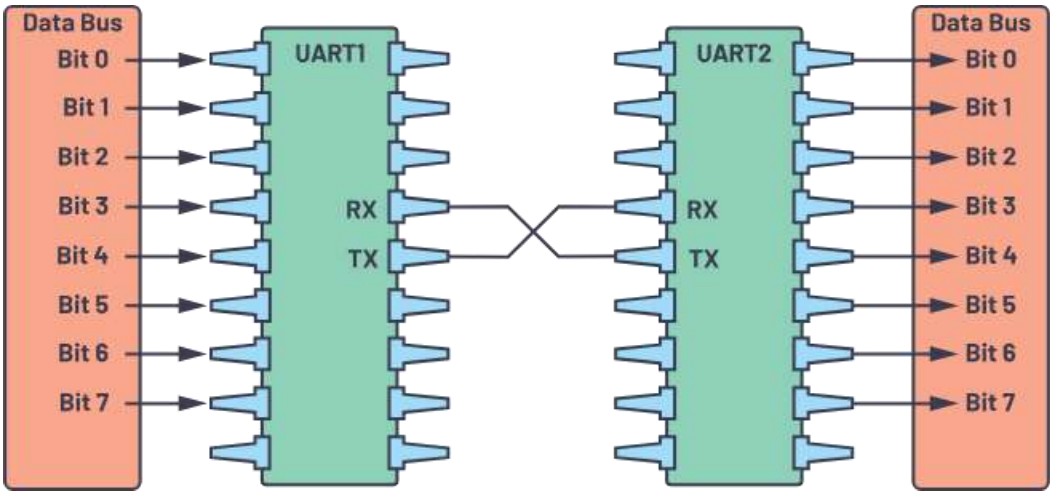


Fig 1: Connection Between 2 Devices

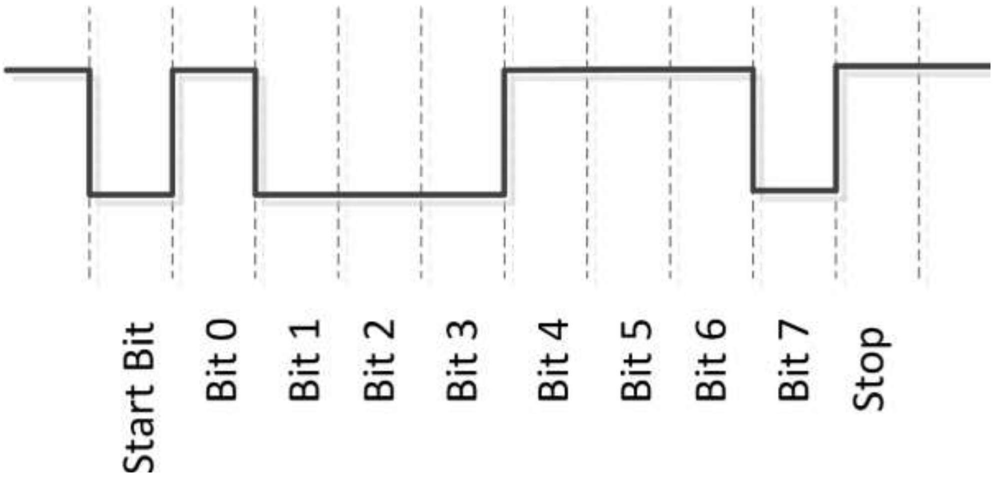


Fig 2: Data Framing

# CHAPTER 3

# LITERATURE REVIEW

*Yamini R. and Ramya M. V. (2020)* presented the design and verification of a full-duplex UART using SystemVerilog. The work integrated transmitter, receiver, baud-rate generator, and FIFO modules under a reusable verification environment consisting of driver, monitor, sequencer, and scoreboard. Randomized test sequences and SystemVerilog Assertions (SVA) were used to achieve 100% functional and assertion coverage. The study effectively reduced verification complexity and time, although it lacked FPGA-based hardware validation.

*Vikash Prajapati and Alpana Pandey (2025)* developed a multi-UART architecture on the Nexys 4 DDR FPGA platform with support for configurable baud rates ranging from 9600 to 230400 bps. The design incorporated transmitter, receiver, and debouncing logic verified through a SystemVerilog testbench using assertions and functional coverage. The study achieved full verification coverage and demonstrated successful hardware implementation, though it was limited to a single FPGA and did not include power optimization.

*Sravani S. and Srujana R. (2023)* designed and verified an autoconfigurable UART controller using Verilog HDL with SystemVerilog-based verification. The system automatically adjusted parameters such as baud rate, data length, and parity without manual reprogramming, enhancing design flexibility. Assertion-based verification ensured data integrity and synchronization, but the work lacked FPGA hardware validation and comprehensive power/resource analysis.

*Neeraj Kumar et al. (2022)* proposed a layered UVM-based verification approach for UART protocol verification. The architecture employed constrained-random test generation, assertions, and coverage analysis to ensure thorough functional validation. The study demonstrated modular and reusable verification infrastructure, but its complexity and longer setup time limited its practicality for simple UART modules.

*Dinesh R. and Aruna Devi P. (2021)* implemented and simulated a UART architecture on FPGA using Verilog HDL. The design was synthesized and tested on Spartan-6 FPGA to verify data transmission reliability through ModelSim simulation. While it provided practical FPGA-level validation, the study lacked advanced constrained-random verification and assertion-based checking methods, focusing mainly on basic functional validation.

## Inference from Literature Review

From the reviewed literature, it is observed that significant progress has been made in the design and verification of UART communication systems using HDL-based methodologies. Existing works have demonstrated efficient UART architectures capable of reliable serial communication with optimized performance in terms of timing accuracy and data integrity. However, most implementations focused primarily on functional correctness, with limited emphasis on systematic verification environments. The reviewed papers highlight the need for a structured verification approach to ensure design robustness and error-free data transmission. Hence, this project aims to bridge this gap by developing a UART communication protocol along with a SystemVerilog-based testbench using UVM-like architecture to achieve comprehensive functional verification and enhanced design reliability.

# CHAPTER 4

# PROPOSED WORK

The proposed work focuses on designing a Universal Asynchronous Receiver Transmitter (UART) protocol and verifying its functionality using a SystemVerilog-based verification environment. The project aims to ensure reliable serial communication through modular design and a structured testbench architecture inspired by the Universal Verification Methodology (UVM) principles.

1. **UART Design Implementation:**
   The UART design consists of two main modules — Transmitter and Receiver — connected through a shared SystemVerilog interface (uart_if).
   - The Transmitter module serializes 8-bit parallel data with start and stop bits and transmits through TxD.
   - The Receiver module deserializes incoming serial data from TxD and reconstructs the original byte, validating data integrity with sampling and synchronization logic.
     Both modules are parameterized with baud rate and clock frequency for scalability.

2. **Verification Architecture:**
   A layered testbench environment is developed to validate UART functionality using object-oriented verification.
   - **Generator:** Creates randomized UART transactions and sends them via mailbox (gen2drv).
   - **Driver:** Drives DUT inputs (TxData, transmit) based on transactions received.
   - **Monitor:** Observes DUT outputs (RxData, valid_rx) and forwards captured data to the scoreboard.
   - **Scoreboard:** Compares expected and received data to ensure accuracy, displaying PASS/FAIL messages.
   - **Environment:** Integrates all components and ensures synchronized operation using mailboxes for inter-process communication.

3. **Testbench-DUT Integration:**

   The UART interface (uart_if) bridges the DUT and the testbench, ensuring signal-level connectivity for clock, reset, transmit, and receive operations. The Uart_Interface module instantiates both the transmitter and receiver to form the DUT.

4. **Simulation and Output Analysis:**

   The testbench runs multiple UART transactions representing the string **“SAURAV”**. The waveform (VCD file) and console outputs verify the correctness of transmission and reception.

   - **Waveform:** Displays bit-level serial transmission timing.
   - **Command Window Output:** Shows transaction flow (GEN, MON, SCB) and functional verification messages.

5. **Outcome and Future Scope:**

   The proposed verification framework demonstrates the correctness of UART communication under simulation. The environment is reusable, modular, and can be extended to test corner cases, error detection mechanisms, and higher-level serial interfaces like **SPI or I2C** in future work.

# CHAPTER 5
# UART CODE

## 5.1 DESIGN CODE:

```
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ uart_design.sv\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

///////////////////In/Out Interface ///////////////////

interface uart_if ;
   logic clk;
   logic reset;
   logic transmit;
   logic [7:0] TxData;
   logic TxD;
   logic busy;
   logic [7:0] RxData;
   logic valid_rx;
endinterface

///////////////////Main Module ///////////////////

module Uart_Interface (uart_if uart);
   Uart_Transmitter tx_inst (uart);
   Uart_Receiver rx_inst (uart);
endmodule

///////////////////Transmitter Module ///////////////////

module Uart_Transmitter (uart_if uart);

parameter int clk_freq = 50_000_000;
parameter int baud_rate = 115200;
parameter int div_counter = clk_freq / baud_rate;

logic [8:0] cycle_counter;
logic [3:0] bit_index;
logic [9:0] tx_shift;
logic sending;

always_ff @(posedge uart.clk or posedge uart.reset) begin
   if (uart.reset) begin
      uart.TxD <= 1;
      sending <= 0;
      cycle_counter <= 0;
      bit_index <= 0;
      uart.busy <= 0;
   end else begin
      if (uart.transmit && !sending) begin
         tx_shift <= {1'b1, uart.TxData, 1'b0};
         sending <= 1;
```

```
          bit_index <= 0;
          cycle_counter <= 0;
          uart.busy <= 1;

      end else if (sending) begin
          if (cycle_counter == div_counter - 1) begin
              cycle_counter <= 0;
              uart.TxD <= tx_shift[bit_index];
              bit_index <= bit_index + 1;
              if (bit_index == 9) begin
                  uart.TxD <= 1;
                  uart.busy <= 0;
                  sending <= 0;
              end
          end else begin
              cycle_counter <= cycle_counter + 1;
          end
      end
   end
end

endmodule

///////////////////Receiver Module //////////////////

module Uart_Receiver (uart_if uart);

parameter int clk_freq = 50_000_000;
parameter int baud_rate = 115200;
parameter int div_sample = 16;
parameter int div_counter = clk_freq / (baud_rate * div_sample);
parameter int mid_sample = div_sample / 2;
parameter int total_bits = 10;
logic state, nextstate; logic shift;
logic [3:0] bit_counter;
logic [3:0] sample_counter;
logic [4:0] cycle_counter;
logic [9:0] rxshift_reg;
logic clear_bitcounter, inc_bitcounter;
logic clear_samplecounter, inc_samplecounter;
logic valid_rx_next;

always_ff @(posedge uart.clk or posedge uart.reset) begin
   if (uart.reset) begin
      state <= 0;
      sample_counter <= 0;
      bit_counter <= 0; cycle_counter<=0;
      rxshift_reg <= 10'b11_1111_1111;
      uart.valid_rx <= 0;
   end else begin
     uart.valid_rx<=0;
      if (cycle_counter == div_counter - 1) begin
          cycle_counter <= 0;
          state <= nextstate;


          if (shift)
            rxshift_reg <= {uart.TxD, rxshift_reg[9:1]};
          if (clear_samplecounter)
              sample_counter <= 0;
          if (inc_samplecounter)
              sample_counter <= sample_counter + 1;
          if (clear_bitcounter)
              bit_counter <= 0;
          if (inc_bitcounter)
              bit_counter <= bit_counter + 1;
          if (valid_rx_next)
              uart.valid_rx <= 1;
          end else begin
            cycle_counter<= cycle_counter +1;
      end
   end
end

always_comb begin
   shift = 0;
   clear_samplecounter = 0;
   inc_samplecounter = 0;
   clear_bitcounter = 0;
   inc_bitcounter = 0;
   valid_rx_next = 0;
   nextstate = state;
   case (state)
      0: begin
        if (!uart.TxD) begin
              nextstate = 1;
              clear_bitcounter = 1;
              clear_samplecounter = 1;
        end
      end
      1: begin
          if (sample_counter == mid_sample)
              shift = 1;
          if (sample_counter == div_sample - 1) begin
              clear_samplecounter = 1;
          if (bit_counter == total_bits - 1) begin
            if (rxshift_reg[9])                               valid_rx_next = 1;
            nextstate = 0;
          end else begin
              inc_bitcounter = 1;
          end
        end else begin
              inc_samplecounter = 1;
        end
      end
   endcase
```

```
end
assign uart.RxData = rxshift_reg[8:1];

endmodule
```

## 5.2 Testbench Code:

```
  \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ uart_testbench.sv\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

///////////////////Main Testbench Module ///////////////////

`timescale 1ns/1ps
`include "uart_transaction.sv"
`include "uart_generator.sv"
`include "uart_driver.sv"
`include "uart_monitor.sv"
`include "uart_scoreboard.sv"
`include "uart_environment.sv"
`include "uart_design.sv"

module tb_uart_interface;
uart_if uart();
Uart_Interface dut (.uart(uart));

// Clock generation
initial uart.clk = 0;
always #10 uart.clk = ~uart.clk; // 50 MHz

environment env;
initial begin
  $dumpfile("uart.vcd");
  $dumpvars(0, tb_uart_interface);
  uart.reset = 1;
  uart.transmit = 0;
  uart.TxData = 0;
  #100
  uart.reset = 0; #100;
  env = new(uart);
  env.run();
  $display("[TB] Simulation completed.");
  $finish;
end
endmodule


      \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ uart_transaction.sv \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

class uart_transaction;
  rand bit [7:0] data;
  bit [7:0] received_data;
  function void display(string tag);
    $display("[%s] Data: %s (%0h)", tag, data, data);
  endfunction
endclass


        \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ uart_generator.sv \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

class generator;
  mailbox gen2drv;

  function new(mailbox gen2drv);
    this.gen2drv = gen2drv;
  endfunction

  task run();
    uart_transaction tr;
    byte message[6] = {"S", "A", "U", "R", "A", "v"};
    for (int i = 0; i < $size(message); i++) begin
      tr = new();
      tr.data = message[i];
      tr.display("GEN");
      gen2drv.put(tr);
      #1000;
    end
  endtask
endclass

          \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ uart_driver.sv \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

class driver;
  virtual uart_if uart;
  mailbox gen2drv;

  function new(virtual uart_if uart, mailbox gen2drv);
    this.uart = uart;
    this.gen2drv = gen2drv;
  endfunction

  task run();
    uart_transaction tr;
    repeat(6) begin
      gen2drv.get(tr);
      @(negedge uart.clk);
      while (uart.busy) @(negedge uart.clk);
      uart.TxData = tr.data;
      uart.transmit = 1;
      @(negedge uart.clk);
      uart.transmit = 0;
    end
  endtask
endclass
```

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ uart_monitor.sv \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

```
class monitor;
  virtual uart_if uart;
  mailbox mon2scb;

  function new(virtual uart_if uart, mailbox mon2scb);
    this.uart = uart;
    this.mon2scb = mon2scb;
  endfunction

  task run();
    int count = 0;
    while (count < 6) begin
      @(posedge uart.clk);
      if (uart.valid_rx) begin
        uart_transaction tr = new();
        tr.received_data = uart.RxData;
        $display("[MON] Received: %s (%0h)", tr.received_data, tr.received_data);
        mon2scb.put(tr);
        count++;
      end
    end
  endtask
endclass
```

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ uart_scoreboard.sv \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

```
class scoreboard;
  mailbox mon2scb;

  function new(mailbox mon2scb);
    this.mon2scb = mon2scb;
  endfunction

  task run();
    uart_transaction tr;
    byte expected[6] = {"S", "A", "U", "R", "A", "V"};
    for (int i = 0; i < $size(expected); i++) begin
      mon2scb.get(tr);
      if (tr.received_data == expected[i]) begin
        $display("[SCB][PASS] Expected %s, Got %s", expected[i], tr.received_data); end
      else begin
        $display("[SCB][FAIL] Expected %s, Got %s", expected[i], tr.received_data);end
    end
  endtask
endclass
```

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ uart_environment.sv \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

```
class environment;
  generator g;
  driver d;
  monitor m;
  scoreboard s;

  mailbox gen2drv = new();
  mailbox mon2scb = new();
  virtual uart_if uart;

  function new(virtual uart_if uart);
    this.uart = uart;
    g = new(gen2drv);
    d = new(uart, gen2drv);
    m = new(uart, mon2scb);
    s = new(mon2scb);
  endfunction

  task run();
    fork
      g.run();
      d.run();
      m.run();
      s.run();
    join
  endtask
endclass
```

# CHAPTER 6

# OUTPUT

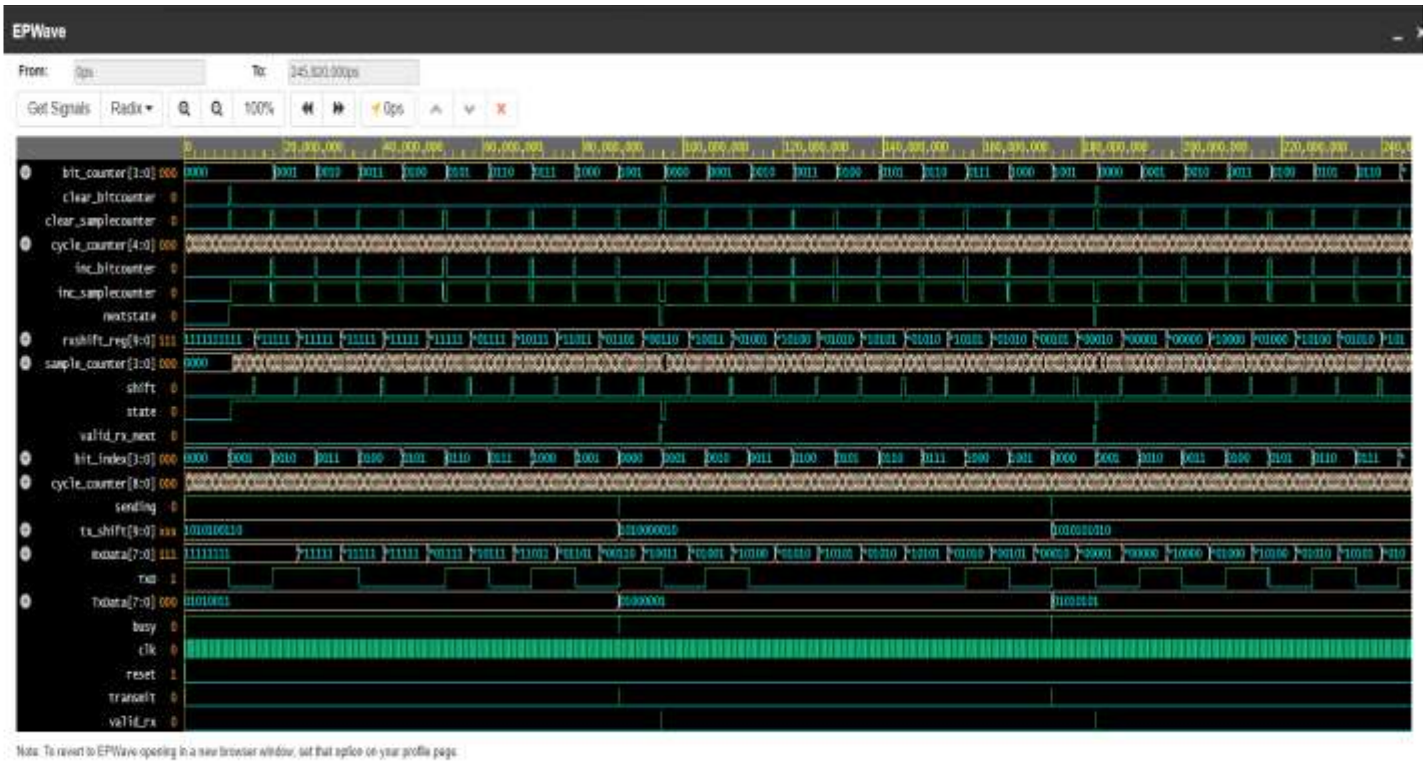
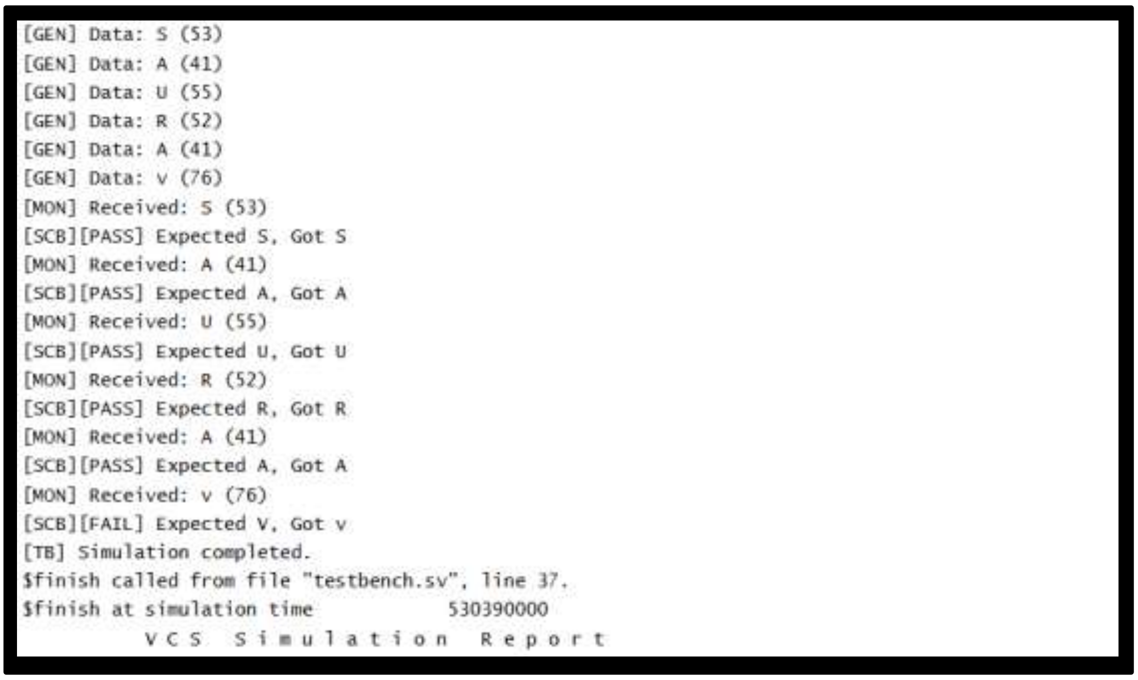


Fig 3: Waveform



```
[GEN] Data: S (53)
[GEN] Data: A (41)
[GEN] Data: U (55)
[GEN] Data: R (52)
[GEN] Data: A (41)
[GEN] Data: v (76)
[MON] Received: S (53)
[SCB][PASS] Expected S, Got S
[MON] Received: A (41)
[SCB][PASS] Expected A, Got A
[MON] Received: U (55)
[SCB][PASS] Expected U, Got U
[MON] Received: R (52)
[SCB][PASS] Expected R, Got R
[MON] Received: A (41)
[SCB][PASS] Expected A, Got A
[MON] Received: v (76)
[SCB][FAIL] Expected V, Got v
[TB] Simulation completed.
$finish called from file "testbench.sv", line 37.
$finish at simulation time           530390000
           V C S   S i m u l a t i o n   R e p o r t
```

Fig 4: Output

# CHAPTER 7
# REFERENCES

1. Wei Ni, Xiaotian Wang, “Functional Coverage-Driven UVM-based UART IP Verification”, IEEE 11th International Conference on ASIC (ASICON), 21 July 2016.

2. Kumari Amrita, Avantika Kumari, “Design And Verification Of Uart Using Verilog Hdl”, International conference on Recent innovations in Management, Engineering, Science and Technology, RIMEST, 2018.

3. Spear, Chris, Tumbush, Greg, “SystemVerilog for Verification” Book.

4. Doron Bustan, Dmitry Korchemny, Erik Seligman, Jin Yang, “SystemVerilog Assertions: Past, Present, and Future SVA Standardization Experience”, IEEE Design & Test of Computers, Volume: 29 , Issue: 2 , April 2012.

5. Renduchinthala H H S S Prasad, Ch. Santhi Rani, ”UART IP Core Verification by using UVM”, International Journal of Industrial Electronics and Electrical Engineering, ISSN: 2347-6982, Volume-4, Issue-7, Jul.-2016

6. Mahat, Nennie Farina. "Design of a 9-bit UART module based on Verilog HDL." Semiconductor Electronics (ICSE), 2012 10th IEEE International Conference on 2012.

7. Ambika, Prof. Anuradha S, “High Speed UART Design Using Verilog”, International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 2, February 2016.